算法基础

什么是算法？

- 算法(algorithm)的**非形式定义**：算法是任何**良定义(well-defined)**的**计算过程**，该过程取某个值或值的集合作为输入，并产生某个值或值的集合作为输出。换句话说，算法是**一个计算步骤的序列**，这些步骤将输入数据转换为输出结果。

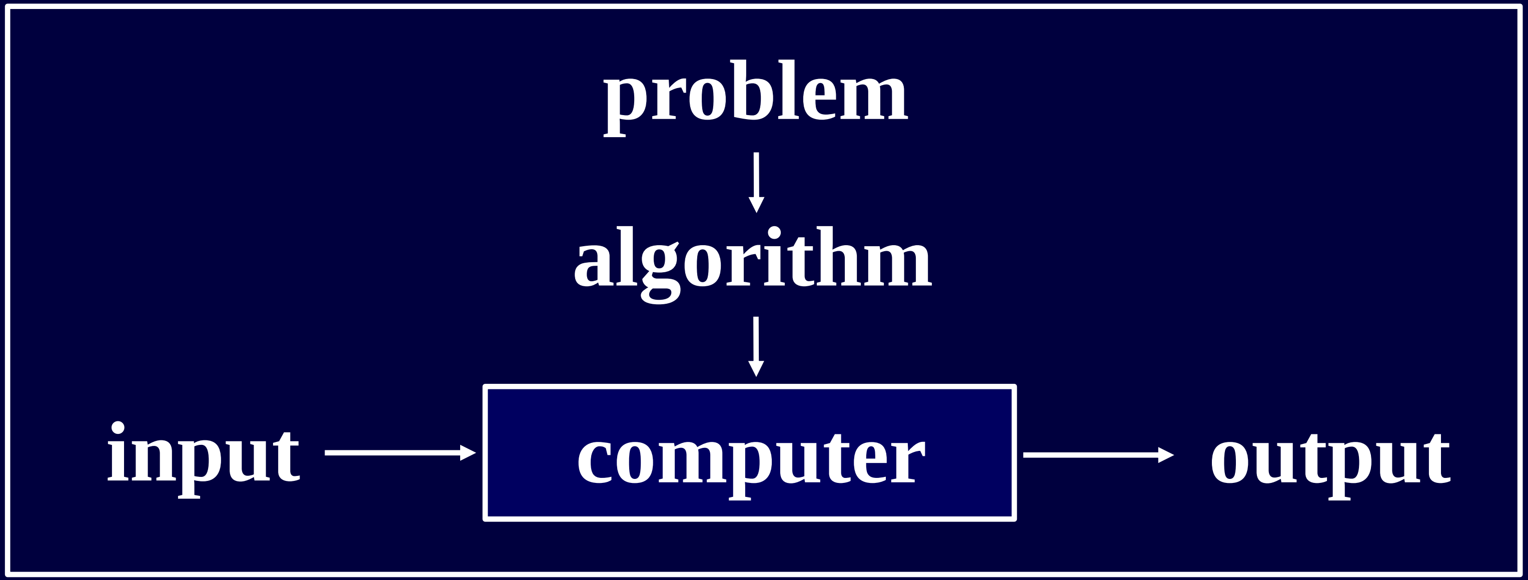
input → algorithm → output

计算步骤

或者说，算法是描述**怎样达到所期望的I/O关系**的计算过程。

什么是算法？(续)

- 算法(algorithm)的另一种定义：一个算法是用于解决一个问题的无歧义指令的序列，该执行序列在有限时间内可获得任何合法的输入所需的输出。



算法相关概念

- 问题(problem)：规定了输入和输出之间的关系，可用通用语言描述。
  - 排序问题：将一系列数按照非降顺序进行排序。

    输入(input)　：具有n个数的数列 $< a_1, a_2, \ldots a_n >$

    输出(output)：对输入数列的一个排列 $< a_1', a_2', \ldots, a_n' >$，

    使得 $a_1' \leq a_2' \leq \cdots \leq a_n'$

- 问题实例：一个问题的实例由计算该问题解所需要的所有输入组成。
  - 排序问题的一个实例：

    input：<12, 17, 67, 41>　——　output：<12, 17, 41, 67>

算法相关概念 (续)

- **输入实例**：问题的具体计算例子
  - 排序问题的**输入实例**：

（1）1, 2, 3, 4, 5, 6, 7.

（2）12, 42, 18, 91.

（3）93, 41, 0, 52, -1, 19, 53, 90, 34.

- **问题规模**：算法的输入实例大小
  - 上述输入实例的**规模**分别为7、4、9。

算法相关概念 (续)

- 正确的算法：若一个算法对问题的每个输入实例，均能终止于正确的输出，则称算法是正确的。

- 不正确的算法：
  - 对某些输入实例不停机；
  - 停机时给出的不是预期的结果。

Note：不正确的算法也并非绝对无用，在不正确的算法的错误概率可控时，该算法有时是有用的 (如大素数算法)。

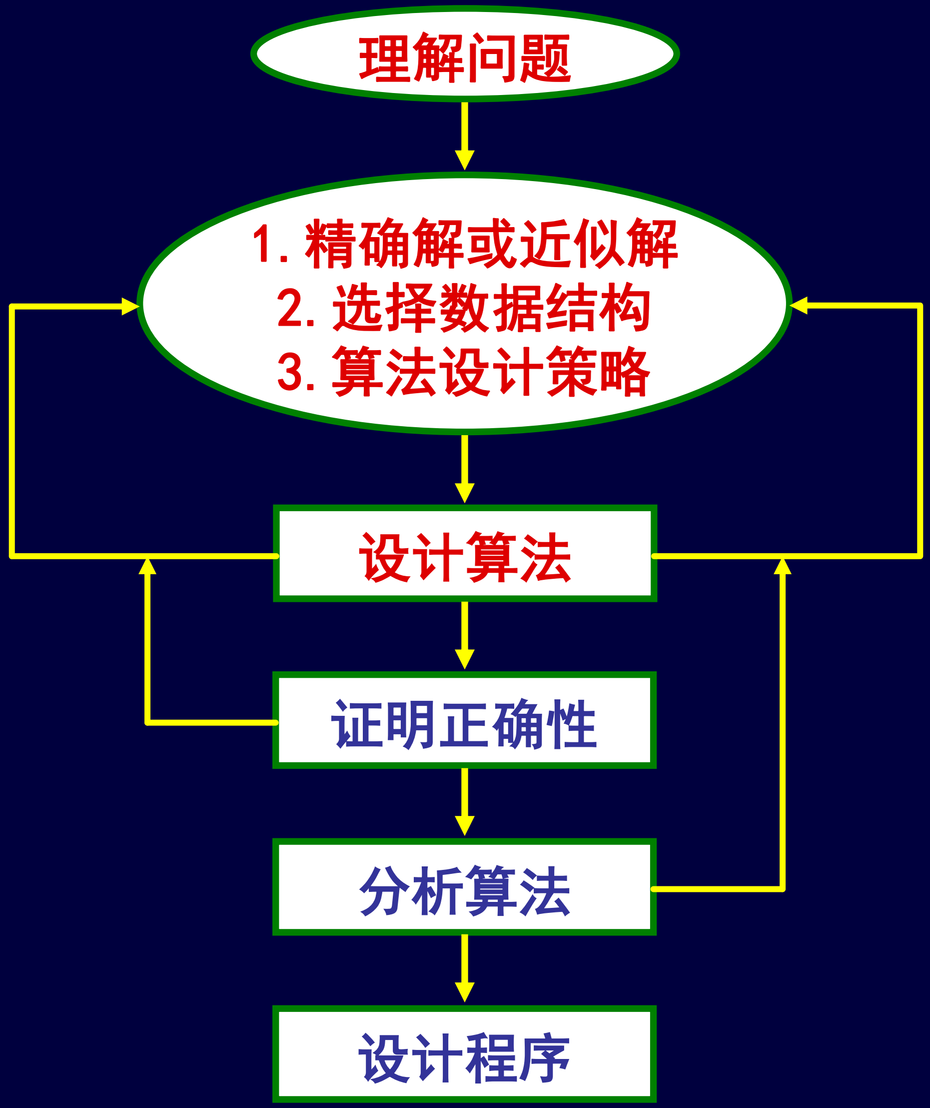
算法相关特征

- 算法(algorithm)的**特征：**
  - **输入：** 一个算法具有零或多个取自执行集合的输入值；
  - **输出：** 对每一次输入，算法具有一或多个与输入值相联系的输出值；
  - **确定性：** 算法的每一个指令步骤都是明确的；
  - **有限性：** 对每一次输入，算法都必须在有限步骤或有限时间内结束；
  - **正确性：** 对每一次输入，算法产生出正确的输出值；
  - **通用性：** 算法的执行过程可应用于所有同类问题，而不仅仅适用于特殊的输入。

算法与程序区别

- 算法(algorithm)：若干指令的有穷序列，满足输入、输出、确定性、有限性和正确性的性质。
- 程序(program)：算法用某种程序设计语言的具体实现，可以不具有有限性。
  - 操作系统是一个在无限循环中执行的程序，因而不是一个算法；
  - 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

问题求解(Problem Solving)过程



算法的描述(Description)

- 算法的描述：可以用英语说明，可以是程序语言，只要能精确描述计算过程即可。
- 伪代码：
  - 拥有自然语言和类编程语言特性，常被用于算法描述；
  - 相较于真实代码的优势：
    1. 对特定代码的描述更加准确清晰；
    2. 不拘泥于技术细节；
    3. 体现算法本质，不受编程语言限制。

算法示例(1) —— Euclid’s algorithm

问题: 寻找两个正整数m和n的最大公约数$gcd(m,n)$

Examples: $gcd(60,0)=60$, $gcd(60,24)=12$, $gcd(m,n)=?$

欧几里得算法(Euclid’s algorithm)是基于对下列等式的反复应用：

$$gcd(m,n)=gcd(n,m \bmod n),$$

当$gcd(m,n)$的第二项变为0，其第一项就成为将输出的结果。

Example: $gcd(60,24)=gcd(24,12)=gcd(12,0)=12$

算法示例(1) —— Euclid's algorithm (续)

- **第一种描述**

  Step 1 If $n = 0$, return m and stop; otherwise go to Step 2.

  Step 2 Divide m by n and assign the value for the remainder to r.

  Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

- **第二种描述**

  $Euclid(m, n)$

  $while\ n \neq 0\ do$

  $\quad r \leftarrow m\ mod\ n$

  $\quad m \leftarrow n$

  $\quad n \leftarrow r$

  $return\ m$

算法示例(2) —— Sieve of Eratosthenes

问题: 找出指定范围$(0, n]$内的所有素数$prime(n)$

Examples: $prime(1) = \{\}$, $prime(5) = \{2, 3, 5\}$, $prime(n) = ?$

The Description of The Sieve of Eratosthenes

$for\ p \leftarrow 2\ to\ n\ do\ A[p] \leftarrow p$
$for\ p \leftarrow 2\ to\ \lfloor\sqrt{n}\rfloor\ do$
    $if\ A[p] \neq 0$ // p hasn't been previously eliminated from the list
        $j \leftarrow p^2$
        $while\ j \leq n\ do$
            $A[j] \leftarrow 0$ // mark element as eliminated
            $j \leftarrow j + p$

Examples: $prime(20) = \{2, 3, 5, 7, 11, 13, 17, 19\}$

算法的重要性(Importance)

- 问题：对10,000,000个整数排序
  - Case 1
    算法 1：插入排序($T(n) = 2n^2$)
    计算机 A：每秒执行10^9条指令(1GHz)
  - Case 2
    算法 2：归并排序($T(n) = 50nlgn$)
    计算机 B：每秒执行10^8条指令(100MHz)
  - 耗时比较
    - Case 1: $\dfrac{2\times(10^7)^2 instructions}{10^9 instructions/s} = 200000s \approx 55.6h$
    - Case 2: $\dfrac{50\times10^7\times log10^7 instructions}{10^8 instructions/s} \approx 116s$

算法分析

定义及目的

- 算法分析：估计算法需要资源的程度。
- 目的1：选择算法
  - 对同一个问题可以设计不同的算法，它们有不同的时间和空间需求；
  - 分析算法可以了解算法的性能，通过比较不同算法的性能，可以选择好的算法避免人力和物力浪费。
- 目的2：优化算法
  - 分析算法可以更加深入了解算法，发现其中的缺陷与不足，从而可以进一步优化算法。

计算模型

- **单处理器计算模型：随机访问机(random-access machine: RAM)**
  - 指令顺序执行，没有并发操作；
  - 包含常用指令，每条指令执行时间为常量；
  - 数据包含整数类型和浮点实数类型；
  - 不对存储器层次进行建模。

- **Note：默认情况下，算法分析一般是指分析算法的时间效率。**

时间分析

- 相关因素
  - 算法的执行时间与输入实例的规模以及输入实例的构成相关。
- 编制不同的数据配置，分析算法的最好、最坏、平均工作情况
  - 最好运行时间：Bogus假象；
  - 最坏运行时间：任何输入实例的运行时间的上界；
  - 平均运行时间：需要对输入数据的分布做出假设。
- 一般考察算法的最坏运行时间
  - 对某些算法，最坏情况经常出现 (比如在数据库中搜索一个不存在的记录)；
  - 对很多算法，平均情况往往与最坏情况大致一样；
  - 若平均时间和最坏时间不是同数量级，那么算法选择依据是：最好、最坏的概率较小时，尽量选择平均时间较小的算法。

时间分析 (续)

- 算法运行时间
  - ①用基本操作的数目(执行步数)来度量，将任何基本操作看作花费单位时间，算法分析独立于机器；
  - ②用更接近实际的计算机上实现的时间来度量，不同的指令具有不同的执行时间，如RAM模型；
  - ①与②之间相差一个常数因子。

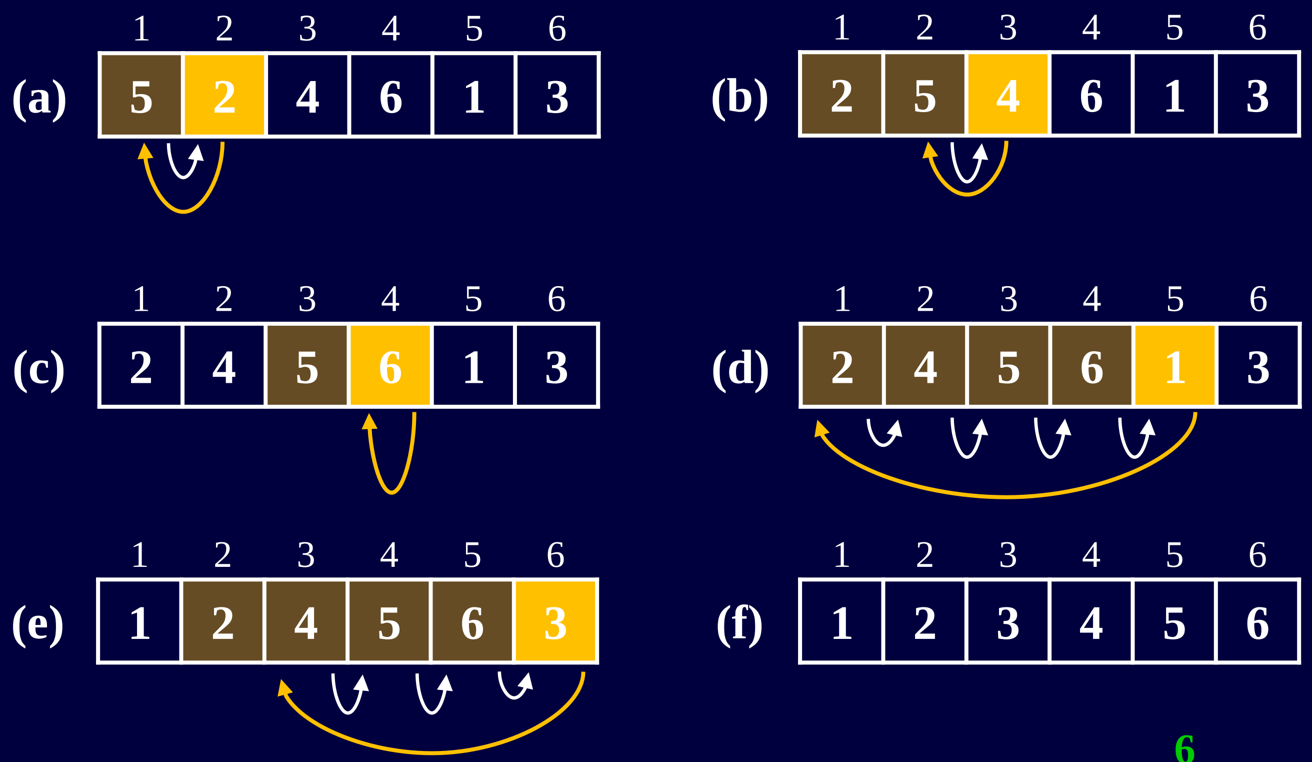
分析示例 —— 插入排序

■ 排序问题：

- ❖ 输入(Input) ： $< a_1, a_2, \ldots, a_n >$
- ❖ 输出(Output)： $< a_1', a_2', \ldots, a_n' >$

■ 插入排序(Insertion Sorting)：

```
INSERTION_SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```



分析示例 —— 插入排序 (续)

- **时间效率分析：**

$INSERTION\_SORT(A)$	cost	times
$for\ j = 2\ to\ A.length$	c_1	n
$\quad key = A[j]$	c_2	$n-1$
$\quad$ // Insert $A[j]$ into the sorted sequence $A[1..j-1]$	0	$n-1$
$\quad i = j - 1$	c_4	$n-1$
$\quad while\ i > 0\ and\ A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$\qquad A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$\qquad i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$\quad A[i+1] = key$	c_8	$n-1$

- **总运行时间：** $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$

分析示例 —— 插入排序 (续)

- 插入排序的运行时间分析：

  - 最好情况：初始数组有序

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \quad \Rightarrow \quad \boldsymbol{g(n) = n}$$

(n的线性函数)

  - 最坏情况：初始数组反向有序

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n(n+1)/2 - 1) + c_6(n(n-1)/2) + c_7(n(n-1)/2) + c_8(n-1)$$

$$= (c_5/2 + c_6/2 + c_7/2)n^2 + (c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8)n - (c_2 + c_4 + c_5 + c_8) \quad \Rightarrow \quad \boldsymbol{g(n) = n^2}$$

(n的二次函数)

算法分析方法

- 在分析插入排序的运行时间时，我们
  - 忽略精确运行时间中的**精确常量(exact constants)**；
  - 忽略精确运行时间中的**低阶项(low order terms)**。
- 我们关注**总运行时间**表达式中的**最高次项**，并**忽略常系数**
  - 当问题规模n足够大时，$T(n)$中的低阶项和常系数不如最高次项中的因子重要；
  - 当问题规模n较小时，$T(n)$中的低阶项和常系数的重要性不可忽略；
  - 需要对时间效率进行“**渐进分析**”。

渐进时间确界

渐进增长

- 在插入排序的示例中，我们讨论了在分析算法时，我们
  - 关注算法的最坏运行时间 (输入规模n的函数)；
  - 忽略精确运行时间中的精确常量 (exact constants)；
  - 忽略精确运行时间中的低阶项 (lower order terms)。
- 什么是插入排序的最坏运行时间？
  - 这取决于我们计算机的速度：
    - 在相同的机器上？
    - 在不同的机器上？
- BIG IDEA:

  --忽略机器相关的常数；

  --关注$n \to \infty$时，$T(n)$的变化情况。

  “渐近分析”

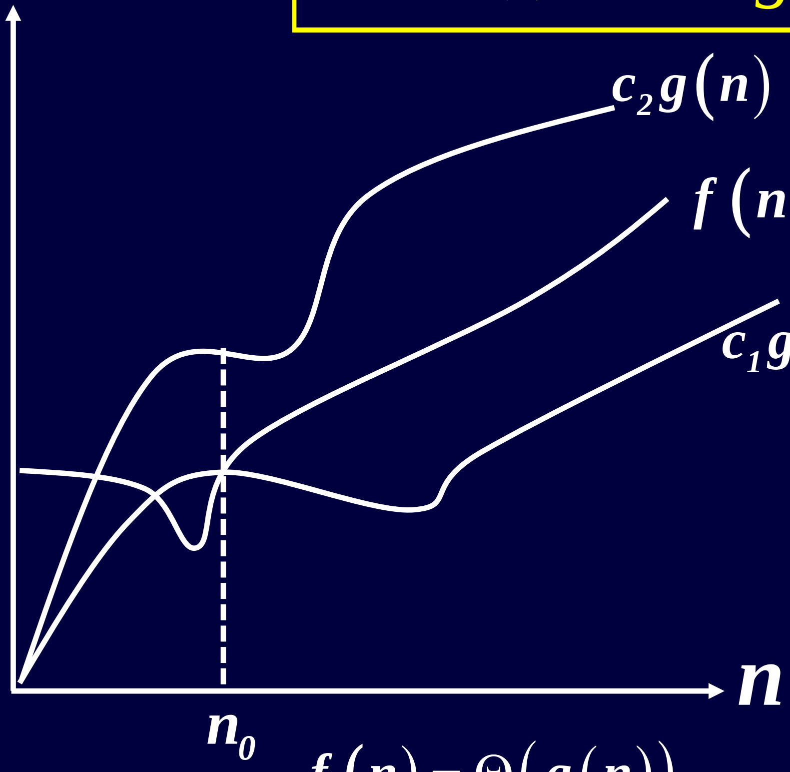
渐进表示

- 算法的渐进时间定义为一个函数，定义域为自然数集合N={0,1,2,…}，但有时也将其扩展到实数或限制到自然数的某个子集上。
- 我们用渐进记号来刻画算法运行的时间，但是渐进记号也可以适用于刻画算法的某个其他方面 (例如算法使用的空间)，甚至可以适用于和算法没有任何关系的函数。

Θ记号（渐进时间确界）

- 定义：给定一个函数g(n)，$\Theta(g(n))$表示一个函数的集合：

$$\boxed{\begin{aligned}&\Theta(g(n))=\{f(n)\mid \exists 常数 c_1, c_2, n_0>0, 使得对所有的\\ &n\ge n_0 有:0\le c_1 g(n)\le f(n)\le c_2 g(n)\} // n为size\end{aligned}}$$



$f(n)=\Theta(g(n))$

- 存在正常数c_1和c_2以及足够大的n，函数$f(n)$能够被“夹在”$c_1g(n)$和$c_2g(n)$之间，则$f(n)$属于集合$\Theta(g(n))$， $f(n)=\Theta(g(n))$。
- 因为$\Theta(g(n))$是一个集合，也可以记为$f(n)\in\Theta(g(n))$。
- 我们称$g(n)$是$f(n)$的一个渐进紧致(确)界(asymptotically tight bound) 。

渐进时间确界 (续)

- 意义：对所有的$n \geq n_0$，函数$f(n)$在一个常数因子范围内等于$g(n)$
- $g(n)$是$f(n)$的一个渐进时间确界，即$g(n)$是$f(n)$是渐进上界和渐进下界
  - $f(n)$ — 算法的计算时间
  - $g(n)$ — 算法时间的数量级
  - 不同的输入实例，一个算法的计算时间$f(n)$不一定相同，故算法的计算时间应该是一个函数集合。

* Note： Θ定义中要求$f(n)$和$g(n)$是渐进非负的 (n足够大时函数值非负)，

否则，$\Theta(g(n))$是空集。

渐进时间确界(续)

- 例：$T(n)=an^2+bn+c$，则$T(n)=\Theta(n^2)$
  - 取$c_1=\frac{a}{4}$，$c_2=7\frac{a}{4}$，$n_0=2max(\frac{|b|}{a},\sqrt{\frac{|c|}{a}})$
  - 则对所有$n\geq n_0$，有$0\leq c_1n^2\leq an^2+bn+c\leq c_2n^2$成立
- 记号$\Theta(1)$：
  - 表示算法的运行时间与问题规模n无关；
  - 亦可理解为常值函数$\Theta(n^0)$：任何常数是0次多项式。

渐进时间确界(续)

- 例：证明 $\frac{1}{2}n(n-1) \in \Theta(n^2)$
  - 若上式成立，则存在正常量c_1，c_2和n_0，使得对所有$n \geq n_0$，有

$$0 \leq c_1 n^2 \leq \frac{1}{2}n(n-1) \leq c_2 n^2$$

  - 当$n \geq 0$时，$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2$，所以可取$c_2 = \frac{1}{2}$;
  - 当$n \geq 2$时，$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n \cdot \frac{1}{2}n \geq \frac{1}{4}n^2$，所以可取$c_1 = \frac{1}{4}$;
  - 综上，取$n_0 = 2$，$c_1 = \frac{1}{4}$，$c_2 = \frac{1}{2}$，有$0 \leq c_1 n^2 \leq \frac{1}{2}n(n-1) \leq c_2 n^2$。

渐进时间确界(续)

- 例：证明 $6n^3 \neq \Theta(n^2)$
  - 可采用反证法。若上式成立，则存在正常量c_1，c_2和n_0，使得对所有$n \geq 0$，有

$$0 \leq c_1 n^2 \leq 6n^3 \leq c_2 n^2$$

  - 直觉告诉我们，问题可能出现在右侧不等式。为了验证这种直觉，我们先证明对任意$n \geq n_0$，有$6n^3 \leq c_2 n^2$。
  - 不等式两边同除以n^2，得到$6n \leq c_2$，即$n \leq \frac{c_2}{6}$，显然不成立
  - 综上，$6n^3 \neq \Theta(n^2)$。

等式中的渐进记号

- 用于替换等式中的低阶项以简化表达式
- 例如：
  - $4n^3 + 3n^2 + 2n + 1 = 4n^3 + 3n^2 + \Theta(n)$

    $= 4n^3 + \Theta(n^2) = \Theta(n^3)$
- 或者，我们可以这样做：
  - $4n^3 + 3n^2 + 2n + 1 = 4n^3 + f(n^2)$，其中我们利用$f(n^2)$简化了该等式。

渐近时间上/下界

O记号（渐近上界）

- Def：对给定函数$g(n)$, $O(g(n))$是一个函数集合

$$\boxed{\begin{aligned} O(g(n)) = \{ f(n) \mid & \exists \text{常数} c, n_0 > 0, \text{such that} \\ & 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \} \end{aligned}}$$

- 即在一个常数因子范围内$g(n)$是 $f(n)$的**渐近上界**
- 注意： $f(n) = \Theta(g(n))$蕴含$f(n) = O(g(n))$，因为**Θ是一个比O记号更强**的概念

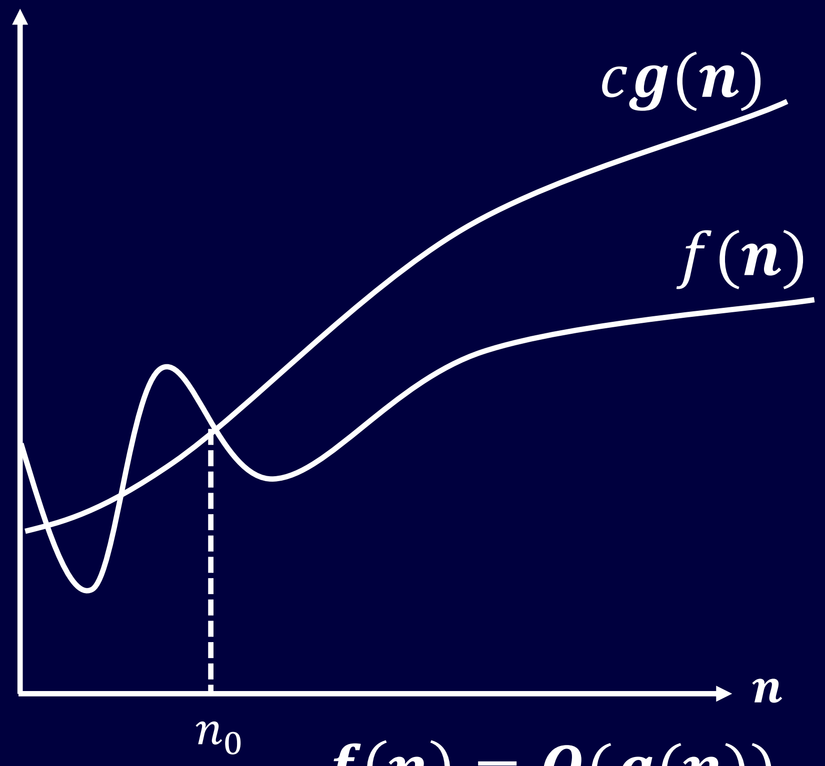
集合论角度：$\because \Theta(g(n)) \subseteq O(g(n))$

$$\therefore f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$$

O记号（渐近上界）

■ 大O的数学定义(渐近上界)

若$g(n)$和$f(n)$是定义在正整数集合上的两个函数，则$f(n)=O(g(n))$表示存在两个正的常数c和n_0，使得当$n \geq n_0$时都满足$0 \leq f(n) \leq cg(n)$。



$f(n)=O(g(n))$

- 函数$f(n)$是集合$O(g(n))$的成员;
- 我们称$g(n)$是$f(n)$的一个**渐近上界**(Asymptotically upper bound)，限制**算法最坏情况运行时间**。

- 大O记号和Θ记号的区别
  - O描述上界，当被用于界定一个最坏运行时间时，蕴含着该算法在任意输入上的运行时间都囿于此界
  - Θ则不然，一个算法的最坏运行时间是$\Theta(g(n))$，并非蕴含着该算法对每个输入实例的运行时间均囿于$\Theta(g(n))$
- 示例：插入排序
  - 在最坏情况下，数组完全逆序，算法的时间复杂度是$O(n^2)$，该界适用于算法对于每个输入的运行时间
  - 但插入排序最坏情况运行时间的界$\Theta(n^2)$并不表示算法对所有输入的运行时间的界也是$\Theta(n^2)$：当输入初始有序时，插入排序的运行时间是$\Theta(n)$，而不是$\Theta(n^2)$。

■ 示例

$$an^3 + bn^2 + cn + d = O(n^3), a > 0$$

证明：$an^3 + bn^2 + cn + d = an^3 + bn^2 + \Theta(n)$

$$= an^3 + \Theta(n^2)$$

$$= \Theta(n^3)$$

又由于集合$\Theta(n^3)$是被$O(n^3)$包含的，故得证

示例（续）

- $\frac{1}{3}n^2 - 3n \in O(n^2)$

  证明：令$\frac{1}{3}n^2 - 3n \le cn^2$，则有$c \ge \frac{1}{3} - \frac{3}{\mathrm{n}}$，当$c = \frac{1}{3}$且$n > 1$时上式成立

- $k_1 n^2 + k_2 n + k_3 \in O(n^2)$

  证明：令$k_1 n^2 + k_2 n + k_3 \le (k_1 + |k_2| + |k_3|)n^2$，

  则当$c > k_1 + |k_2| + |k_3|$且$n \ge 1$时，$k_1 n^2 + k_2 n + k_3 \le cn^2$成立

- $k_1 n^2 + k_2 n + k_3 \in O(n^3)$

  证明：$k_1 n^2 + k_2 n + k_3 \le (k_1 + |k_2| + |k_3|)n^3$

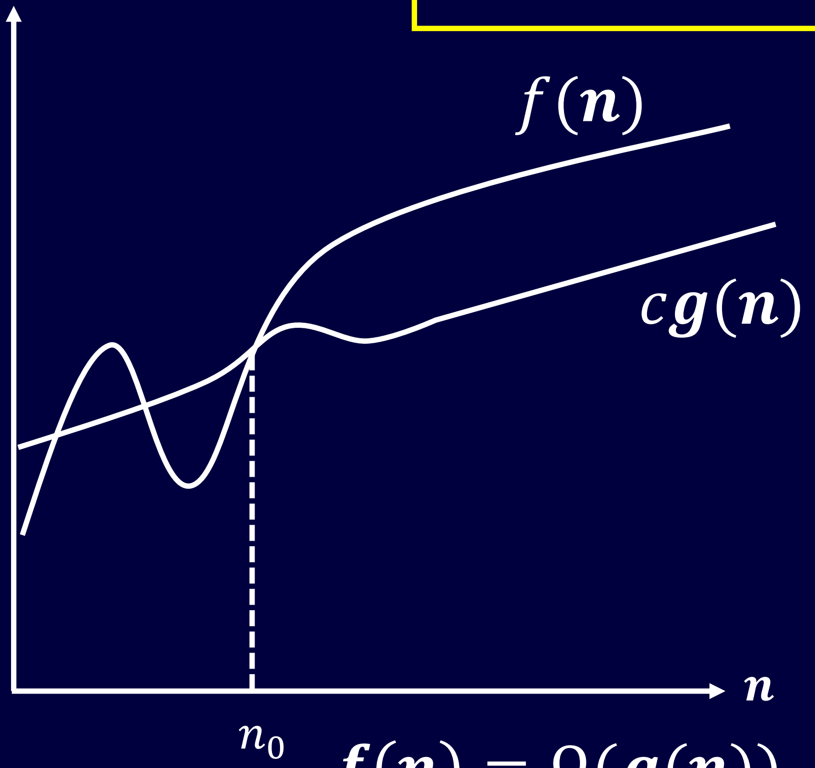
O记号使用时的注意事项

- 当说算法的运行时间上界是$O(n^2)$，往往是指其最坏运行时间，无须修饰语，对那些最好、最坏、平均时间数量级不同者均成立，而Θ则要分开表达、加修饰语。
- 使用大O表示法通常可以更容易地分析算法；比如我们可以很容易地证明插入排序的运行时间上界是$O(n^2)$。
- 一些非形式化描述：
  - 我们常用$f(n) = O(g(n))$来替代$f(n) \in O(g(n))$
  - 我们常在等式中使用$O(n)$：$2n^2 + 3n + 1 = 2n^2 + O(n)$表示$2n^2 + 3n + 1 = 2n^2 + f(n)$，其中$f(n) \in O(n)$，可消去不必要的细节，突出主项的常数因子等。
  - 常函数被写作$O(1)$

Ω记号（渐近下界）

- ***Def*** **: 对给定函数$g(n)$, $\Omega(g(n))$是一个函数集合：**

$$\boxed{\begin{aligned}\Omega(g(n)) = \{ f(n) \mid &\exists 常数c,\ n_0 > 0\ such\ that \\ &0 \le cg(n) \le f(n)\ for\ all\ n \ge n_0 \}\end{aligned}}$$



$f(n) = \Omega(g(n))$

- 存在正常量c使得对于n_0及其右边的所有n值，函数$f(n)$值总大于$cg(n)$，则$f(n) \in \boldsymbol{\Omega}(\boldsymbol{g}(\boldsymbol{n}))$
- 我们称 $g(n)$ 是 $f(n)$ 的 **渐近下界** (Asymptotically lower bound)

Ω记号（渐近下界）

- **Theorem. 1**：对任意函数$f(n)$和$g(n)$，$f(n) = \Theta(g(n))$当且仅当$f(n) \in O(g(n))$和$f(n) \in \Omega(g(n))$。

  即：$g(n)$是$f(n)$的渐紧界当且仅当$g(n)$是$f(n)$的渐近上界和渐近下界

- 当Ω用来界定一个算法的最好情况下的运行时间时，蕴含着该算法在任意输入上的运行时间都囿于此界。

- 示例：插入排序的下界是$\Omega(n)$，即对任何实例成立，插入排序的最好运行时间是$\Omega(n)$

  $$\because n = \Omega(n)\ n^2 = \Omega(n)$$

o记号（渐近非紧确上界）

- 大O记号表示的渐近上界可以是渐近紧致的，也可以是渐近非紧界

$$2n^2 = O\left(n^2\right) \because \frac{2n^2}{n^2} \rightarrow 2\text{，紧致界}$$

$$2n = O\left(n^2\right) \because \frac{2n}{n^2} \rightarrow 0\text{，非紧致界}$$

- 小o记号用来表示——函数的渐近非紧致上界

找出在形式化定义上与大O记号的两处差别

- *Def*：$$o(g(n)) = \{ f(n) \mid \forall 常数c > 0, \exists 常数n_0 > 0 \ such \ that \ 0 \le f(n) < cg(n) \text{ for all } n \ge n_0 \}$$

只要n足够大，$g(n)$是$f(n)$的上界。

例：$2n = o(n^2)$ 但 $2n^2 \neq o(n^2)$

直观上，当$n \to \infty$，$f(n)$相对于$g(n)$是可忽略的。

即：$f(n) = o(g(n))$蕴含着$\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = 0$

或者说f和g数量级不同，否则不可能对于任意常数c，都有$cg(n)$严格大于$f(n)$。

ω记号（渐近非紧确下界）

找出在形式化定义上与大Ω记号的两处差别

- **Def:** $\omega(g(n)) = \{ f(n) \mid \forall 常数c > 0, \exists 常数n_0 > 0 \ such\ that\ 0 \le cg(n) < f(n) \text{ for all } n \ge n_0 \}$

即：对任意常数$c > 0$，$cg(n)$对足够大的n要严格小于$f(n)$。

$\therefore f$和g必定不是同数量级，(f量级$>$ g量级)

- 例：$\frac{n^2}{2} = \omega(n)$ 但 $\frac{n^2}{2} \ne \omega(n^2)$

$$f(n) = \omega(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

函数间的比较

比较各种函数

- 许多实数的关系性质可用(引申)到渐近比较，下面假定$f(n)$和$g(n)$是渐近正的
  - 传递性 (对于五种渐进记号均适用)

$$f(n)=\Theta(g(n)) \text{ and } g(n)=\Theta(h(n)) \Rightarrow f(n)=\Theta(h(n)) \text{ //渐紧界}$$
$$f(n)=O(g(n)) \text{ and } g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$$
$$f(n)=\Omega(g(n)) \text{ and } g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$$
$$f(n)=o(g(n)) \text{ and } g(n)=o(h(n)) \Rightarrow f(n)=o(h(n)) \text{ //渐近非紧上界}$$
$$f(n)=\omega(g(n)) \text{ and } g(n)=\omega(h(n)) \Rightarrow f(n)=\omega(h(n)) \text{ //渐近非紧下界}$$

  - 自反性

$$f(n)=\Theta(f(n)),\quad f(n)=O(f(n)), f(n)=\Omega(f(n))$$

渐近非紧界无自反性

比较各种函数（续）

- **对称性 (仅对渐近紧确界成立)**

$$f(n)=\Theta(g(n))\ iff\ \ g(n)=\Theta(f(n))$$

紧致界有对称性

- **转置对称性**

$$f(n)=O(g(n))\ iff\ \ g(n)=\Omega(f(n))$$

//大O与大Ω对调时，f、g对称

$$f(n)=o(g(n))\ iff\ \ g(n)=\omega(f(n))$$

//g是f的非紧上界等价于f是g的非紧下界

比较各种函数（续）

- 由上述4个性质，可将两函数间的渐近比较类比于两个实数间的比较。

$$f(n)=O(g(n)) \approx a \le b \quad //\text{“}\approx\text{”相似于，} f \,\square\, a, g \,\square\, b$$

$$f(n)=\Omega(g(n)) \approx a \ge b$$

$$f(n)=\Theta(g(n)) \approx a = b$$

$$f(n)=o(g(n)) \approx a < b \quad // f(n)\text{渐近小于} g(n)$$

$$f(n)=\omega(g(n)) \approx a > b \quad // f(n)\text{渐近大于} g(n)$$

但实数的三岐性(三分性质)不能类比到渐近表示中：

三岐性：$\forall a, b \in R$，下述三种情况必有一个成立：

$$a < b, a = b, or\ a > b$$

即任意两实数间是可比较的

比较各种函数（续）

Theorem. 1：对任意函数 $f(n)$ 和 $g(n)$, $f(n)=\Theta(g(n))$ 当且仅当 $f(n)=O(g(n))$ 和 $f(n)=\Omega(g(n))$

- **并非所有函数都是渐近可比较的**

  即$\exists f(n)$和$g(n)$,

  可能$f(n)=O(g(n))$不成立,

  而$f(n)=\Omega(g(n))$也不成立,

  则由**Theorem.1**知，$f(n)\neq\Theta(g(n))$

- **例：函数n和$n^{1+\sin n}$之间是无法渐近比较的**

  $\because \mathbf{1+\sin n \in [0,2]}$

  $\therefore \mathbf{n^{1+\sin n}}$在$\mathbf{O(1)\sim O(n^2)}$之间波动

函数比较示例

- **例1. 独立增长双函数情况**

证明：如果$t_1(n)\in O(f(n))$且$t_2(n)\in O(g(n))$，

则$t_1(n)+t_2(n)\in O(\max\{f(n),g(n)\})$.

$Proof: \because t_1(n)\in O(f(n))$,

$\therefore \exists$正常量c_1和非负整数n_1，对所有$n\geq n_1$,

s. t. $t_1(n)\leq c_1 f(n)$

同理 $\exists$正常量c_2和非负整数n_2，对所有$n\geq n_2$,

s. t. $t_2(n)\leq c_2 g(n)$

函数比较示例（续）

- **例1. 独立增长双函数情况（续）**

证明：如果$t_1(n)\in O(f(n))$且$t_2(n)\in O(g(n))$,

则$t_1(n)+t_2(n)\in O(\max\{f(n),g(n)\})$.

令$\mathrm{c}_3=\max\{c_1,c_2\}$，$\mathrm{n}_3\geq\max\{n_1,n_2\}$

$$
\begin{aligned}
t_1(n)+t_2(n) &\leq c_1 f(n)+c_2 g(n)\\
&\leq \mathrm{c}_3 f(n)+\mathrm{c}_3 g(n)\\
&= \mathrm{c}_3\left[f(n)+g(n)\right]\\
&\leq 2\mathrm{c}_3\max\{f(n),g(n)\}
\end{aligned}
$$

故$t_1(n)+t_2(n)\in O(\max\{f(n),g(n)\})$,

其中$c=2\max\{c_1,c_2\}$，$n_0=\max\{n_1,n_2\}$，得证

函数比较示例（续）

■ 例1. 独立增长双函数情况（续）

上述结论对于Ω和Θ是否同样成立？

注：尽管符号O, Ω, Θ的正式定义对于记住它们的抽象性质必不可少，但我们很少直接用定义来比较两个特定函数的增长次数。

一种简便的方法是对两个函数的比值求极限：

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & f(n)\text{增长率小于}g(n) \quad ① \\ c>0 & f(n)\text{增长率等于}g(n) \quad ② \\ \infty & f(n)\text{增长率大于}g(n) \quad ③ \\ \text{不存在} & \quad ④\end{cases} \Rightarrow$$

①&②： $f(n)\in O(g(n))$

②&③： $f(n)\in \Omega(g(n))$

②： $f(n)\in \Theta(g(n))$

④： 该方法不适用

函数比较示例（续）

- **例2. 利用比值法求证** $\log_2 n \in o(\sqrt{n})$

$$Proof: \lim_{n\to\infty}\frac{\log_2 n}{\sqrt{n}} = \lim_{n\to\infty}\frac{(\log_2 n)'}{(\sqrt{n})'} = \lim_{n\to\infty}\frac{(\log_2 e)\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = 2\log_2 e\lim_{n\to\infty}\frac{1}{\sqrt{n}} = 0$$

$\therefore \log_2 n$增长率（阶数）低于$\sqrt{n}$，故得证

注：借助洛必达法则，可以推广至n^{ε}

函数比较示例（续）

- 算数运算 (请同学课下证明)
  - $O(f(n)) + O(g(n)) = O(f(n) + g(n))$
  - $O(f(n)) * O(g(n)) = O(f(n) * g(n))$
  - $O(cf(n)) = O(f(n))$
  - $g(n) = O\big(f(n)\big) \Rightarrow O\big(f(n)\big) + O\big(g(n)\big) = O\big(f(n)\big)$

渐近分析法总结

- 一种忽略常数因子和输入小项的函数比较法

  - $O(g(n))$: class of functions $f(n)$ that grow <u>no faster</u> than $g(n)$
  - $\Theta(g(n))$: class of functions $f(n)$ that grow <u>at same rate</u> as $g(n)$
  - $\Omega(g(n))$: class of functions $f(n)$ that grow <u>at least as fast</u> as $g(n)$